wildcards

BASICS OF THE UNIX/LINUX ENVIRONMENT

UNIX is a four letter word

''Unix is user friendly --
It's just picky about who it's friends are...''
-- Unknown, seen in .sigs around the world

Introduction to wildcards.

Wildcards are essential when dealing with almost anything in terms of text processing. (Looking for/Managing files from the command line is text processing.)

They are a subset of regular expressions, an essential (i.e. esoteric and difficult) Unix feature.

Introduction to wildcards.

Example

Say I want to find all the files in the working directory that begin with the letter "<u>a</u>".

(lower case only since Unix is case sensitive.)

Start out with the ls command

```
%ls a???
```

How do we specify we want all combinations of all characters following the "a"?
We use a wildcard.

```
%ls a*
```

The asterisk "*" (called "splat") wildcard means match a string with any number of any characters (including none, so will match a file "a").

Try it ---

```
alpaca.ceri.memphis.edu/rsmalley 143:> ls a*
a.out                           antex.sh
antarctic sun panorama 3x.ai    atantest.f
antarctic sun panorama.125.jpg  awk
antarctic sun panorama.25.jpg   az_map
antarctic sun panorama.ai       az_map.ps
antarctic sun panorama.jpg

adelitst:
aadeli.ini      adelitst.sh     jessai          pessai
ADELI.MESSAGES  eessai          kcnusc.pal      PLOT1
ADELI.MINMAX    iessai          oessai          tempi

arc2gmtstuff:
arcgmt.README  arcgmt.tar    arcgmt_ai     arcgmt_av
alpaca.ceri.memphis.edu/rsmalley 144:>
```

(As part of the <u>regular expression</u> feature of Unix) wildcards can be used in combination with almost all Unix commands.

Wildcards

"*" - matches zero or more characters or numbers.

Find all files in local subdirectory SEIS with the string "<u>.BHZ.</u>" in their file name.

```
%ls   SEIS/*.BHZ.*
SEIS/HIA.BHZ.SAC           SEIS/WMQ.BHZ.SAC
SEIS/filt.HIA.BHZ.SAC       SEIS/filt.WMQ.BHZ.SAC
```

Wildcards

"*" - asterisk - matches zero or more characters or numbers.

Combining/multiple use of wildcards.

Find all files in local subdirectory SEIS that begin with the letter "f" and also have the string ".BHZ." in their file name.

```
%ls SEIS/f*.BHZ.*
SEIS/filt.HIA.BHZ.SAC      SEIS/filt.WMQ.BHZ.SAC
```

“?” - question mark - matches a single character or number.

Find all files in local subdirectory SEIS that have the name “HIA.BH” plus some single letter (the ?) plus a “.” and then plus anything (the *).

```
% ls  SEIS/HIA.BH?.*
SEIS/HIA.BHE.SAC          SEIS/HIA.BHN.SAC
SEIS/HIA.BHZ.SAC
```

Wildcards

"[ ]" - brackets - used to specify a set or range of characters or numbers rather than all possible characters or numbers.

Find all files in local subdirectory SEIS that have the name "HIA.BH" plus one of E, N or Z (the stuff in brackets) plus a "." and then plus anything (the *).

```
%ls   SEIS/HIA.BH[E,N,Z].*
SEIS/HIA.BHE.SAC           SEIS/HIA.BHZ.SAC
SEIS/HIA.BHN.SAC
```

Wildcards

Find all files in all local subdirectories (the first *) that have the name "<u>HIA</u>" plus <u>anything</u> (the second *) plus the characters "<u>198</u>" plus a <u>single character in the range 0-9</u> then plus <u>anything</u> (the third and last *).

```
%ls   */HIA*198[0-9]*
795/HIA.BHZ.D.1988.041:07.18.30
799/HIA.BHZ.D.1988:14:35:27.00
812/HIA.BHZ.D.1988:03:43:49.00
813/HIA.BHZ.D.1988.362:13.58.59
814/HIA.BHZ.D.1989.041:17.07.43
```

Some random stuff

A note on Control-C (CTRL-C)

Use CTRL-C (hold down Control [CTRL] key, then type "C" and finally release Control key] to quit a job (stop whatever is going on).

If you accidently start something that isn't working, CTRL-C will quit and return you to a blank command line.

Some random stuff

A note on the book

As the book was not written for the CERI system, some of the files it refers to are not located where the book says they are.

What we have seen so far

Commands

----

cd
pwd
ls
mkdir
rmdir
rm
more
less
cat
paste
head
tail
cp
mv
ln

See this link for a list and description of many Unix commands

http://pcsplace.com/tech-list/ultimate-list-of-linux-and-unix-commands/

What we have seen so far

Redirection

Pipes

Switches

Some special characters (~,\,.,..)

Wildcards (*,?)

man pages

BASICS OF THE UNIX/LINUX ENVIRONMENT

Using man pages

Using man pages

Layout

All man pages follow a common layout that is optimized for presentation on a simple ASCII text display, possibly without any form of highlighting or font control.

Using man pages

Typical man page has following "headings":

SECTION
NAME
SYNOPSIS
DESCRIPTION
OPTIONS
OPERANDS
USAGE
(EXAMPLES)
ENVIRONMENT VARIABLES
EXIT STATUS
(FILES)
ATTRIBUTES
SEE ALSO
NOTES
(BUGS)

```
alpaca.ceri.memphis.edu/rsmalley 141:> man ls
Reformatting page.  Please Wait... done

User Commands                                          ls(1)

NAME
     ls - list contents of directory

SYNOPSIS
     /usr/bin/ls [-aAbcCdfFghilLmnopqrRstux1@] [file...]

     /usr/xpg4/bin/ls [-aAbcCdfFghilLmnopqrRstux1@] [file...]

DESCRIPTION
     For each file that is a directory, ls lists the contents  of
     the  directory.  For  each file that is an ordinary file, ls
     repeats its name and any other  information  requested.  The
     output is sorted alphabetically by default. When no argument
     is given, the current  directory  is  listed.  When  several
     arguments   are   given,  the  arguments  are  first  sorted
     appropriately, but file arguments appear before  directories
     and their contents.

     There are three major listing formats.  The  default  format
     for  output  directed  to  a  terminal  is multi-column with
     entries sorted down the columns. The -1 option allows single
     column  output and -m enables stream output format. In order
     to determine output formats for the -C, -x, and -m  options,
     ls  uses  an environment variable, COLUMNS, to determine the
     number of character positions available on one output  line.
     If  this  variable  is  not set, the terminfo(4) database is
     used to determine  the  number  of  columns,  based  on  the
     environment  variable,  TERM.  If this information cannot be
     obtained, 80 columns are assumed.

     The mode printed under the -l option consists of ten charac-
     ters. The first character may be one of the following:
```

SECTION

NAME

SYNOPSIS

DESCRIPTION

Using man pages

SECTION: The section of the manual. Includes command whose man page you requested.

```
User Commands                                                  ls(1)
```

The ls commnad is in the "User Commands" section of the documentation/manual, which is section #1.

NAME: The name of the command or function, followed by a one-line description of what it does.

```
NAME
     ls - list contents of directory
```

Using man pages

SYNOPSIS

In the case of a command, you get a formal description of how to run it and what command line options it takes. For program functions, a list of the parameters the function takes and which header file contains its definition. For experienced users, this may be all the documentation they need.

Using man pages

SYNOPSIS (not so obvious)

Shows where command lives - `/usr/bin/` - (there are 2 versions available, depends on your path - more on paths later), plus ...

```
SYNOPSIS
     /usr/bin/ls [-aAbcCdfFghilLmnopqrRstux1@] [file...]

     /usr/xpg4/bin/ls [-aAbcCdfFghilLmnopqrRstux1@] [file...]
```

Using man pages

SYNOPSIS (not so obvious)

...list of options
{ [-aAbcCdfFghilLmnopqrRstux1@] }
the brackets { [ ] } signify that the stuff inside the brackets is optional, and ...

```
SYNOPSIS
     /usr/bin/ls [-aAbcCdfFghilLmnopqrRstux1@] [file...]

     /usr/xpg4/bin/ls [-aAbcCdfFghilLmnopqrRstux1@] [file...]
```

Using man pages

SYNOPSIS (not so obvious)

… finally, optionally (the brackets) a file name (file), that may be repeated an arbitrary number of times - the ellipses { . . . }.

```
SYNOPSIS
     /usr/bin/ls [-aAbcCdfFghilLmnopqrRstux1@] [file...]

     /usr/xpg4/bin/ls [-aAbcCdfFghilLmnopqrRstux1@] [file...]
```

Using man pages

Brackets - optional parameters.

File - filename.

Ellipses - repeat as necessary.

Using man pages

DESCRIPTION

A textual description of the functioning of the command or function.

Using man pages

DESCRIPTION

The DESCRIPTION can go on for a number of pages.

```
DESCRIPTION
     For each file that is a directory, ls lists the contents  of
     the  directory.  For  each file that is an ordinary file, ls
     repeats its name and any other  information  requested.  The
     output is sorted alphabetically by default. When no argument
     is given, the current  directory  is  listed.  When  several
     arguments   are   given,  the  arguments  are  first  sorted
     appropriately, but file arguments appear before  directories
     and their contents.

     There are three major listing formats.  The  default  format
```

This is where we find out what the first letters of the long ls format mean

The mode printed under the -l option consists of ten characters. The first character may be one of the following:

d	The entry is a directory.
D	The entry is a door.
l	The entry is a symbolic link.
b	The entry is a block special file.
c	The entry is a character special file.
p	The entry is a FIFO (or "named pipe") special file.
s	The entry is an AF_UNIX address family socket.
-	The entry is an ordinary file.

etc.

Using man pages

OPTIONS

Specification of the command's options

```
OPTIONS
     The following options are supported:

     -a     Lists all entries, including those that begin  with  a
            dot (.), which are normally not listed.

     -A     Lists all entries, including those that begin  with  a
            dot  (.),  with the exception of the working directory
            (.) and the parent directory (..).

     -b     Forces printing of non-printable characters to  be  in
            the octal \ddd notation.
```

This can go on for pages also.

Using man pages

OPERAND

Describes the valid operands.

```
OPERANDS
     The following operand is supported:

     file   A path name of a file  to  be  written.  If  the  file
            specified  is  not found, a diagnostic message will be
            output on standard error.
```

Explains the operand is optional file name(s).

Using man pages

USAGE

Notes on usage (not examples).

```
USAGE
     See largefile(5) for the description of the behavior  of  ls
     when encountering files greater than or equal to 2 Gbyte ( 2
    **31 bytes).
```

Using man pages

EXAMPLES

Optionally gives some examples.

```
EXAMPLES
     Example 3: Providing file information
     Another example of a command line is:

     example% ls -aisn

     This command provides information on  all  files,  including
     those  that  begin  with  a dot (a), the i-number-the memory
     address of the i-node associated with  the  file-printed  in
     the left-hand column (i); the size (in blocks) of the files,
     printed in the column to the right  of  the  i-numbers  (s);
     finally,  the  report is displayed in the numeric version of
     the long list, printing the UID (instead of user  name)  and
     GID  (instead  of  group  name)  numbers associated with the
     files.
     When the sizes of the files in a  directory  are  listed,  a
     total   count  of  blocks,  including  indirect  blocks,  is
     printed.
```

Using man pages

Followed by a bunch of other (mostly) esoteric stuff.

ENVIRONMENT VARIABLES (these can get you), EXIT STATUS, FILES, ATTRIBUTES, (the following may be useful) SEE ALSO, NOTES, BUGS.

Shells

BASICS OF THE UNIX/LINUX ENVIRONMENT

What is a shell?

As far as Unix is concerned, the shell is just another program.

As far as the user in concerned, it is the traditional command line user interface with the Unix operating system…it interprets your typing.

What is a shell?

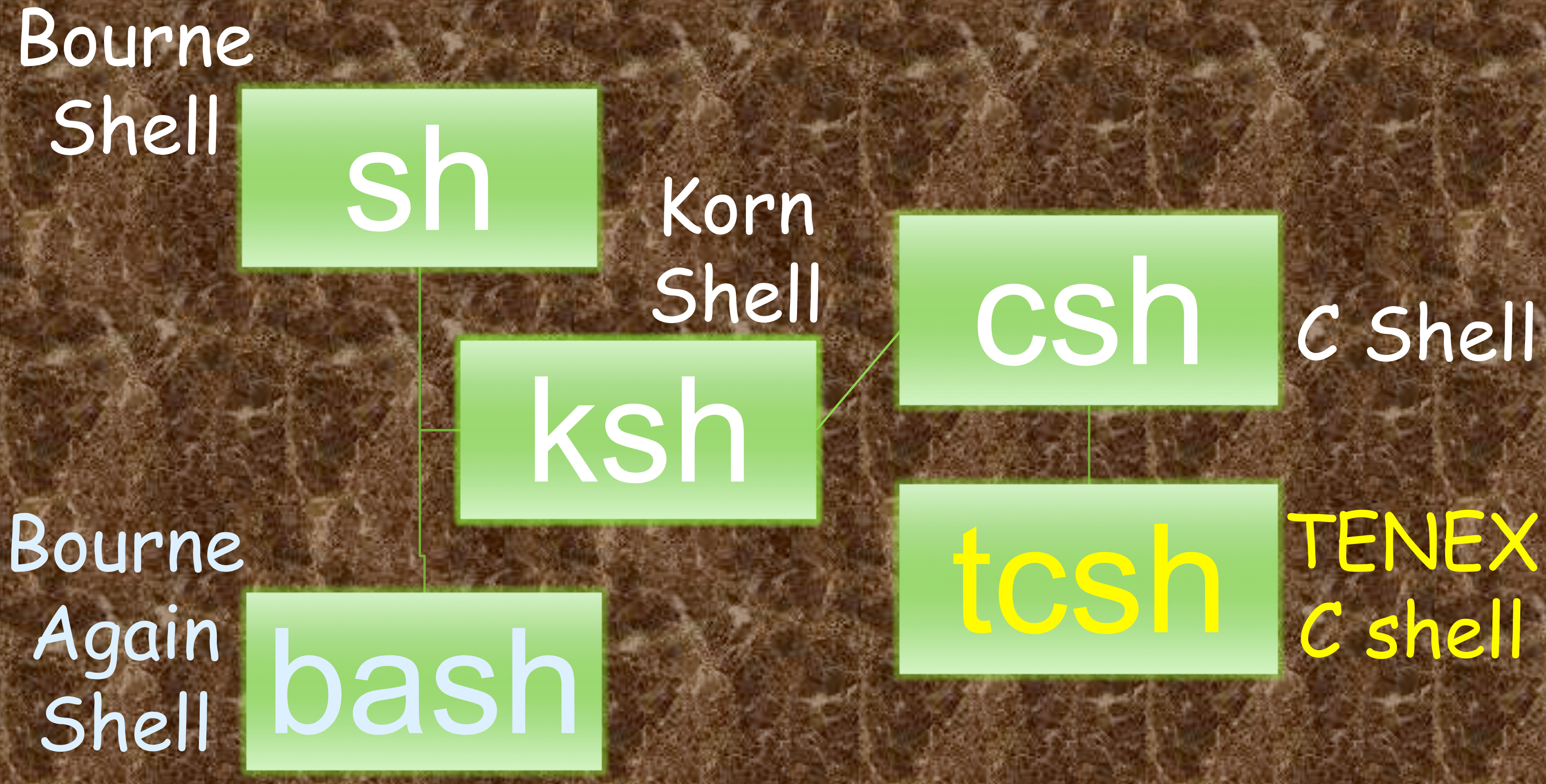
Just as there are many flavors of Unix and Unix-like systems, there are many types of shells.

If you don't like any of the shells in existence, this is Unix – write your own!

Common shells

Bourne Shell	sh
Bourne Again Shell (current default on MAC OS X)	bash
C Shell	csh
TENEX C Shell (This is the default shell at CERI)	tcsh
Korn Shell (mix between two families above)	ksh

Common shells



sh

Bourne shell

The original Unix shell.

Pro: Flexible and powerful scripting shell.

Con: Not interactive or particularly user friendly.

csh

C shell

designed for the BSD Unix system.

syntax closely follows C programming.

Pro: easy for C programmers to learn and comes with many interactive features such as file completion, aliases, history.

Con: not as flexible or powerful a scripting language.

ksh

Korn shell

derived from the Bourne shell so has a shared syntax.

job control taken from the C shell.

bash

Bourne-Again shell

Combines the “best” of sh, ksh, and csh.

Default shell on Linux and Mac OSX operating systems.

Pro: Flexible and powerful scripting language with all the interactive features of csh plus command completion.
This shell is great for complicated GMT scripts.

tcsh

TENEX C shell

Default shell of the CERI unix environment.

Pro: User friendly on the command line.

Con: It is not as suitable for long and involved scripts.

It is perfectly OK for most daily geophysics work on the command line & most faculty here use it on a daily basis so there are many experts around.

What is my shell?

This seems to be the best way to find out.

```
%echo $0
```

Works for tcsh, sh, and bash.

($0 does not refer to the shell in general, this may be one of the Unix “standards” that $0 is the program you are running!!).

What is my shell?

```
alpaca.ceri.memphis.edu/rsmalley 145:> echo $0
/usr/bin/tcsh

alpaca.ceri.memphis.edu/rsmalley 146:> /bin/sh
$ echo $0
/bin/sh

$ /bin/bash
bash-2.05$ echo $0
/bin/bash

bash-2.05$ exit
Exit

$ echo $0
/bin/sh

$ exit
alpaca.ceri.memphis.edu/rsmalley 147:> echo $0
/usr/bin/tcsh
alpaca.ceri.memphis.edu/rsmalley 148:>
```

Query for shell

Run sh
Query for shell

Run bash
Query for shell

Exit bash,
returns to sh

Query for shell

Exit sh, returns
to tcsh

What is my shell?

```
alpaca.ceri.memphis.edu/rsmalley 145:> echo $0
/usr/bin/tcsh

alpaca.ceri.memphis.edu/rsmalley 146:> /bin/sh
$ echo $0
/bin/sh

$ /bin/bash
bash-2.05$ echo $0
/bin/bash

bash-2.05$ exit
Exit

$ echo $0
/bin/sh

$ exit
alpaca.ceri.memphis.edu/rsmalley 147:> echo $0
/usr/bin/tcsh
alpaca.ceri.memphis.edu/rsmalley 148:>
```

Can also id the shell by the prompts (once you know which is which).

These examples also show that shell is just another program – the only thing special about it is that one is started automatically for you when you login.

What is my shell?
The commands

```
%env $SHELL
%echo $SHELL
```

will echo the value of the environment variable $SHELL to the screen - but this may not be your shell!

```
alpaca.ceri.memphis.edu/rsmalley 152:> echo $0
/usr/bin/tcsh
alpaca.ceri.memphis.edu/rsmalley 153:> echo $SHELL
/usr/bin/tcsh
alpaca.ceri.memphis.edu/rsmalley 154:> echo $shell
/usr/bin/tcsh
alpaca.ceri.memphis.edu/rsmalley 155:> /bin/sh
$ echo $SHELL
/usr/bin/tcsh
$ echo $shell

$
```

Useful features of tcsh & bash

-<u>file completion</u>-

you can key the tab key, or the escape key twice, to complete the name of a long file.

Useful features of tcsh & bash

<u>history command</u>

list the previous commands entered during the active session.

```
alpaca.ceri.memphis.edu/rsmalley 149:> history
. . .
   145   21:30    pwd
   146   21:30    DEM
   147   21:30    cd srtm
   148   21:30    history
```

Useful features of tcsh & bash

-history “feature”-

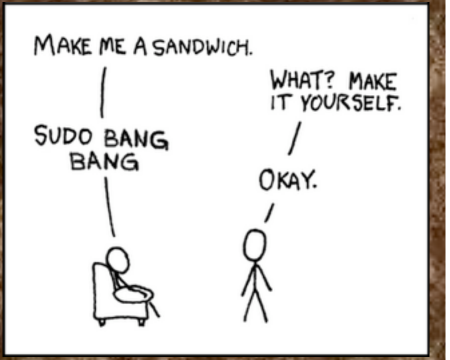
up and down arrow keys: allow you to move up and down through previous commands.

right and left arrow keys: allow you to edit command lines (backspace to remove, type at cursor to insert) without starting from scratch.

Useful features of tcsh & bash

bang ("!") command/shortcut

Bang is used to search backward through your *Bash* history until it finds a command that matches the string that follows it and returns/executes it.



bang ("!") command/shortcut

!XXX<CR> returns the command numbered XXX in the history list, and in this ex. It runs it after you enter the <CR>.)

```
alpaca.ceri.memphis.edu/rsmalley 149:> history
. . .
   145  21:30   pwd
   146  21:30   DEM
   147  21:30   cd srtm
   148  21:30   history
alpaca.ceri.memphis.edu/rsmalley 149:> !146
DEM
/gaia/home/rsmalley/dem
alpaca.ceri.memphis.edu/rsmalley 150:>
```

bang ("!") command

!-X: returns the command X back in the history list and runs it at the <CR>.

```
alpaca.ceri.memphis.edu/rsmalley 151:> history
. . .
   147   21:30    cd srtm
   148   21:30    cd ~
   149   21:30    history
   150   21:46    DEM
   151   21:55    history
alpaca.ceri.memphis.edu/rsmalley 152:> !-4
cd ~
/gaia/home/rsmalley
alpaca.ceri.memphis.edu/rsmalley 153:>
```

bang ("!") command/shortcut

!ca: retuns the last command in the history file beginning with "ca".

!!: retuns the last command in the history list.

bang ("!") command/shortcut is actually more general - use it to return commands from history and do something with them.

For the purposes of these tips, every tip will assume these are the last three commands you ran:

```
% which firefox
% make
% ./foo -f foo.conf
% vi foo.c bar.c
```

Getting stuff from the last command:

Full line: `% !!` becomes: `% vi foo.c bar.c`

Various shells have options that can affect this.

Be careful with shells that let you share history among instances. Some shells also allow bang commands to be expanded with tabs or expanded and reloaded on the command line for further editing when you press return.

bang (“!”) command/shortcut is actually more general - use it to return commands from history and do something with them.

For the purposes of these tips, every tip will assume these are the last three commands you ran:

```
% which firefox
% make
% ./foo -f foo.conf
% vi foo.c bar.c
```

Getting stuff from the last command:

Last arg : `% svn ci !$` becomes: `% svn ci bar.c`

bang ("!") command/shortcut is actually more general - use it to return commands from history and do something with them.

For the purposes of these tips, every tip will assume these are the last three commands you ran:

```
% which firefox
% make
% ./foo -f foo.conf
% vi foo.c bar.c
```

Getting stuff from the last command:

All args : `% svn ci !*` becomes: `% svn ci foo.c bar.c`

bang ("!") command/shortcut is actually more general - use it to return commands from history and do something with them.

For the purposes of these tips, every tip will assume these are the last three commands you ran:

```
% which firefox
% make
% ./foo -f foo.conf
% vi foo.c bar.c
```

Getting stuff from the last command:

First arg: `% svn ci !!:1` becomes: `% svn ci foo.c` bang ("!") command/shortcut is actually more general - use it to return commands from history and do something with them.

The colon is a separator for specifying further options/details of the desired action.

First arg: `% svn ci !!:1` becomes: `% svn ci foo.c`

bang ("!") command/shortcut

For the purposes of these tips, every tip will assume these are the last three commands you ran:

```
% which firefox
% make
% ./foo -f foo.conf
% vi foo.c bar.c
```

We will see what each of these commands (except make) does later.

bang ("!") command/shortcut

For the purposes of these tips, every tip will assume these are the last three commands you ran:

```
% which firefox
% make
% ./foo -f foo.conf
% vi foo.c bar.c
```

Accessing command lines by pattern:

Full line: `% !./f` becomes: `% ./foo -f foo.conf`

bang ("!") command/shortcut

For the purposes of these tips, every tip will assume these are the last three commands you ran:

```
% which firefox
% make
% ./foo -f foo.conf
% vi foo.c bar.c
```

Accessing command lines by pattern:

Full line: `% vi `!whi`` becomes: `% vi `which firefox``

bang ("!") command/shortcut

For the purposes of these tips, every tip will assume these are the last three commands you ran:

```
% which firefox
% make
% ./foo -f foo.conf
% vi foo.c bar.c
```

Accessing command lines by pattern:

All args :

```
% ./bar !./f:*
```

becomes:

```
% ./bar -f foo.conf
```

We are looking for "./f", and then (the colon, ":") want all args (the splat, "*")

bang ("!") command/shortcut

For the purposes of these tips, every tip will assume these are the last three commands you ran:

```
% which firefox
% make
% ./foo -f foo.conf
% vi foo.c bar.c
```

Accessing command lines by pattern:

First arg: `% svn ci !vi:1` becomes: `% svn ci foo.conf`

bang ("!") command/shortcut

Notice how this makes perfect sense under the Unix philosophy.

Make a tool and (mis/ab)use it.

(the basic commands are really very simple, but in tricky combination they become very powerful.)

Most normal people are not going to use all these shortcuts, they are just too complicated.

I showed them, however, to present additional application of the Unix philosophy.

bang ("!") command/shortcut

you can also check the command bang finds before executing it.

```
!cat:p<CR>
```

Now, instead of executing the command it finds, bang prints the command to Standard OUT for you to look at.

bang ("!") command/shortcut

```
!cat:p<CR>
```

That's not all though, it also copies the command to the end of your history (even though it was not executed).

This is useful because if you do want to execute that command, you can now use the *bang bang* shortcut to run it (*bang bang* runs the last thing in history).

bang ("!") command/shortcut

```
!cat:p<CR>
!! | grep "hello"<CR>
```

Here, the most recent command containing *cat* is printed, and copied to the end of your history.

Then, that command is executed with its results being piped into the *grep* command, which has been specified to print those lines containing the string "hello".

bang (“!”) command/shortcut

Ever run a command only to have it fail for lack of superuser privileges?

Instead of retyping the whole command with *sudo* or even pressing the up arrow and scrolling back to the beginning of the command to type *sudo*, you can just type this:

```
sudo !!
```

bang ("!") command/shortcut

To find a lot of this "neat" stuff I just GOOGLEd

"unix bang command"

----

you will not find it in the man pages

```
alpaca.ceri.memphis.edu/rsmalley 147:> man !
No manual entry for !.
alpaca.ceri.memphis.edu/rsmalley 148:>
```

Modify last command in history list using caret or circumflex accent, "^", to fix typos or make small changes.
Replaces text inside first two carets with that between second and third.

(can sometimes skip closing caret as shown below in second example.)

```
smalleys-imac-2:documents smalley$ ls trk1.kml
trk1.kml
smalleys-imac-2:documents smalley$ ^1^2^
ls trk2.kml
trk2.kml
smalleys-imac-2:documents smalley$ !!:p
ls trk2.kml
smalleys-imac-2:documents smalley$
smalleys-imac-2:documents smalley$ ^2^1
ls trk1.kml
trk1.kml
smalleys-imac-2:documents smalley$
```

Environment (esoteric and essential)

BASICS OF THE UNIX/LINUX ENVIRONMENT

The Unix Environment (general and CERI specific)

Mitch has set up the basic CERI <u>environment</u> so that everyone can access the standard Unix tools and geophysics packages available on the Unix system at CERI.

The Unix Environment

But what does this mean?

Many UNIX utilities, including the shell, need information about you and what you're doing in order to do a reasonable job.

What kinds of information?

Well, to start with, a lot of programs (particularly editors) need to know what kind of terminal you're using.

Your environment is composed of a number of *environment variables* which provide this important information to the operating system.

Rather than forcing you to type this information with every command

```
such as (% mail -editor vi -term aardvark48)
```

UNIX uses *environment variables* to store information that you'd rather not worry about.

For example, the *TERM* environment variable tells programs what kind of terminal you're using. Any programs that care about your terminal type know (or ought to know) that they can read this variable, find out your terminal type, and act accordingly.

UNIX commands receive information from three potential sources.

-Arguments on the command line

-Data coming down their standard input channel.

-The *environment*. When a command is started, it is sent a list of *environment variables* by the shell.

Since you generally want the computer to behave the same way everyday, these *environment variables* are setup and stored in configuration files that are accessed automatically at login.

What are your environment variables?

env: prints the current environment variables to the screen.

```
alpaca.ceri.memphis.edu/rsmalley 141:> env
USER=rsmalley
LOGNAME=rsmalley
HOME=/gaia/home/rsmalley
PATH=.:/gaia/home/rsmalley:/gaia/home/rsmalley/bin:/gaia/home/
rsmalley/shells:/gaia/home/rsmalley/dem:/gaia/home/rsmalley/
defm:/gaia/home/rsmalley/defm/src:/gaia/home/rsmalley/
visco1d_pollitz/viscoprogs_rs:/gaia/home/rsmalley/gg:/gaia/
home/rsmalley/gg/com:/gaia/home/rsmalley/gg/gamit/bin:/gaia/
home/rsmalley/gg/kf/bin:/gaia/dunedain/d2/gps/bin:/gaia/
smeagol/local/passcal.2006/bin:/gaia/smeagol/local/gmt/
GMT4.2.1/bin:/usr/sbin:/usr/local/teTeX/bin/sparc-sun-
solaris2.8:/gaia/home/rsmalley/bin:/opt/local/sbin:/opt/sfw/
bin:/usr/bin:/usr/ccs/bin:/usr/local/bin:/opt/SUNWspro/SC5.0/
bin:/opt/local/bin:/usr/bin:/usr/dt/bin:/usr/openwin/bin:/
bin:/usr/ucb:/gaia/smeagol/local/bin:/net/gps4/d1/Noah/rbh/
usr/PROGRAMS.330/bin:/gaia/home/rsmalley/X/bin:/gaia/home/
rsmalley/X/com:/gaia/home/rsmalley/record_reading/bin:/gaia/
home/rsmalley/record_reading/scripts
MAIL=/var/mail//rsmalley
SHELL=/usr/bin/tcsh
TZ=US/Central
LC_CTYPE=en_US.ISO8859-1
LC_COLLATE=en_US.ISO8859-1
```

```
LC_TIME=en_US.ISO8859-1
LC_NUMERIC=en_US.ISO8859-1
LC_MONETARY=en_US.ISO8859-1
LC_MESSAGES=C
SSH_CLIENT=75.66.47.230 50561 22
SSH_CONNECTION=75.66.47.230 50561 141.225.157.63 22
SSH_TTY=/dev/pts/12
TERM=xterm
HOSTTYPE=sun4
VENDOR=sun
OSTYPE=solaris
MACHTYPE=sparc
SHLVL=1
PWD=/gaia/home/rsmalley
GROUP=user
HOST=alpaca.ceri.memphis.edu
REMOTEHOST=c-75-66-47-230.hsd1.tn.comcast.net
MANPATH=/gaia/smeagol/local/passcal.2006/man:/gaia/smeagol/
local/gmt/GMT4.2.1/man:/ceri/local/man:/usr/dt/man:/usr/man:/
usr/openwin/share/man:/usr/local/man:/opt/SUNWspro/man:/opt/
sfw/man:/usr/local/teTeX/man:/gaia/smeagol/local/man
LD_LIBRARY_PATH=/gaia/smeagol/local/gmt/lib:/gaia/opt/
SUNWspro/lib:/gaia/opt/SUNWspro/SC5.0/lib:/usr/lib:/usr/
openwin/lib
```

```
LM_LICENSE_FILE=/gaia/opt/licenses/licenses_combined
GMTHOME=/gaia/smeagol/local/gmt/GMT4.2.1
NETCDFHOME=/gaia/smeagol/local/gmt
GMT_GRIDDIR=/gaia/smeagol/local/gmt/GMT4.2.1/share/dbase
GMT_IMGDIR=/gaia/smeagol/local/gmt/GMT4.2.1/DATA/img
GMT_DATADIR=/gaia/smeagol/local/gmt/GMT4.2.1/DATA/misc
CWD=/gaia/home/rsmalley
EDITOR=vi
AB2_DEFAULTSERVER=http://stilgar.ceri.memphis.edu:8888
PRINTER=3892
```

You get all the stuff shown so far automatically.

```
HELP_DIR=/gaia/home/rsmalley/gg/help/
INSTITUTE=uom
RECORD_READING=/gaia/home/rsmalley/record_reading
RECORD_READING_BIN=/gaia/home/rsmalley/record_reading/bin
RECORD_READING_SCR=/gaia/home/rsmalley/record_reading/scripts
RECORD_READING_SRC=/gaia/home/rsmalley/record_reading/src
latestrtvel=rtvel4_9305_5bv19
LATESTRTVEL=rtvel4_9305_5bv19
ANONFTP=/gaia/midtown/mid4/smalley/public_ftp
ANONFTP_IN=/gaia/midtown/mid4/smalley/public_ftpinbox
SACDIR=/gaia/tesuji/d1/local/sac
SACXWINDOWS=x11
SACAUX=/gaia/tesuji/d1/local/sac/aux
SACSUNWINDOWS=0
GPSHOME=/gaia/dunedain/d2/gps
```

Plus you can add our own stuff (above).

Unless you are running Linux (in which case you are the system manager), you can forget about setting up most of this as the system managers do it for you.

There are a few environment variables, however, that you need to know about and/or set up yourself.

HOME*

This environment variable controls what other Unix commands consider your base or home directory.

This is how "%cd" and "~" know which directory to refer to

```
% echo $HOME
/gaia/home/rsmalley
```

To refer to an environment variable put a $ in front of the name.

The $ therefore has a special meaning to the shell.

(As do the characters " ~, !, /, *, ?, ^ " which we have already seen. By the time we are done we will have used up most of the non alpha-numeric characters with special meanings.)

SHELL*

This variable stores your default shell

```
% echo $SHELL
/usr/bin/tcsh
```

(however this may give an incorrect result.)

*these environment variables should not be changed by the user